Service-Oriented Computing

Aiello, Marco; Dustdar, Schahram

Published in:
EPRINTS-BOOK-TITLE

Publication date:
2006

Service-Oriented Computing: Service Foundations

Marco Aiello and Schahram Dustdar

TUWien
{aiellom,dustdar}@infosys.tuwien.ac.at

Participating in the discussion: Paco Curbera, Flavio De Paoli, Wolfgang Kellerer, Dominik Kuropka, Frank Leymann, Michal Zaremba

1 Service Foundations

The foundations of Service-Oriented Computing are concerned with the precise definition of a service. This is not only about providing a verbal explanation of what a service is, what can be and what cannot be considered a service, but rather it is about identifying the appropriate *service model*. Defining what service properties, requirements, and behaviors are relevant and possible for any generic service.

Any task based on services has to then take into account the service model, in fact, it is shaped by the service model chosen. Think of service composition. In whichever way one wants to consider composition (static ad design-time, dynamic at execution time, etc.) one has to consider what are the basic blocks to build the composition out of. One has to know which properties and behaviors of the service are available to guarantee certain properties and behaviors of the composition.

The remainder of this document is organized as follows. Section 1.1 identifies the architectural peculiarities of the service model. Section 1.2 presents the container model. Section 2 enumerates a number of challenges and open research issues on service foundations. The paper is concluded by Section 3 which is the proposal for a book part on service foundations within a book dealing with state of the art and open issues in Service-Oriented Computing.

1.1 Architectural Principles

A number of basic properties and characteristics define services and must, therefore, be taken into consideration when defining the service model. Let us consider the architectural principles which are the common basis for all services.

Loose coupling Interacting services are loosely coupled by nature. They run on different platforms, are implemented independently and have different owners. The model has to consider the loose coupling of services with respect to one another.

Dagstuhl Seminar Proceedings 05462
Service Oriented Computing (SOC)
http://drops.dagstuhl.de/opus/volltexte/2006/528

Message based interaction The service based interaction is message based. Furthermore, the messages are usually exchanged asynchronously and have a rich document style content. RPC is seldom used for service.

Dynamic discovery Services are programs available over a network, it is thus paramount to have the possibility to find which services are available at any given moment and what the service is able to do. Dynamic service discovery must be supported by the service model. A scenario which highlights the necessity for dynamic discovery is that of mobile service requesters and providers (see Figure 1). In this scenario, there is not full knowledge of available service implementation nor total reachability of all implementations. So services need to be discovered dynamically and on the basis of the given requester's context.

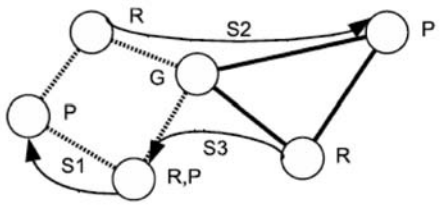

Fig. 1. Mobility scenario for dynamic discovery.

Late binding Services are invoked dynamically after the discovery process by a service requestor. That is, the binding of a service provider to the requester occurs at run time at the latest possible instant providing for an extremely dynamic and flexible architecture.

Implementation neutrality and independency Services are defined independently of their implementation and behave neutrally with respect to it.

Policy based behavior Properties such as transactions, security, and context should not be tightly bound to service implementations but rather defined using policies, rules and other declarative forms.

Configurability Services are defined abstractly and are deployed at a later stage. The configuration of the services is dynamic and can occur at any time of the service's life. The deployment of services must thus account for the highest possible flexibility.

Autonomicity Autonomic deployment of services in large-scale environments may provide for greater ease of service management. Self-healing capabilities are possible and desirable of a service.

Portability Service implementation should be independent from concrete implementation as possible. Portability is a basic characteristic of services (cf. WS-Basic Profile). Today service implementations are portable but concrete deployment is *not* (service deployed at runtime A cannot be deployed at

runtime B without changing deployment descriptors etc.). To make an analogy, this is similar to the migration of EJB implementations between various servers (e.g., IBM, BEA).

Granularity The granularity of a service defines the complexity and number of functionalities defined by an individual service and is thus part of the service model. An appropriate balance between coarse and fine grained services depends on the way services are modeled. For too fine grained services, problems arise when there are frequent and rapid changes.

1.2 The Container Model

One model that is emerging as appropriate and successful for the Service-Oriented Paradigm is the *Container Model*. In this model, there is a 'container' for the service implementation taking care of exposing the service functionalities to the external world via the network. Figure 2 provides a schematization of the container model, where SD stands for the service description interface for functional (e.g., WSDL) and non-functional features (e.g., WS-Security).

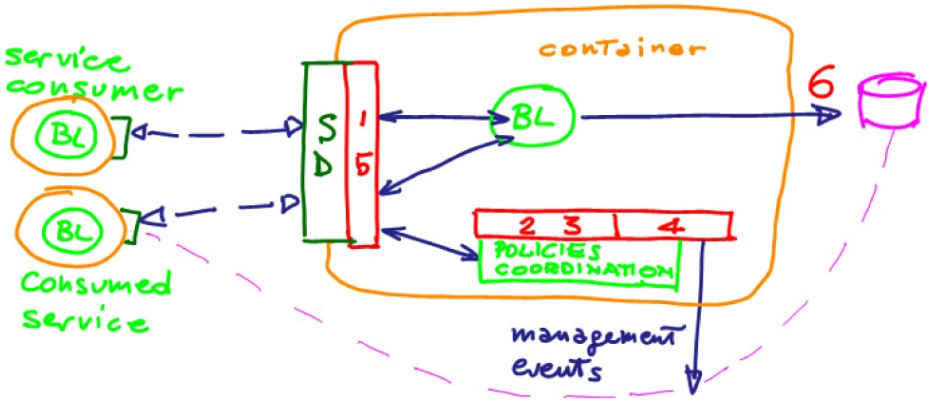

Fig. 2. The container model.

The basic (core) functions of the container are the following six,

1. Connectivity and MEPs (e.g., ESB, Event-brokers, P/S)
2. (Extensible Architecture) Mechanisms to support and provision requirements such as Transactions, Security, Performance metrics etc
3. Support for Dynamic Configuration
4. Monitoring of internal behavior and state to management systems (services)
5. Data and Protocol Adaptation
6. Discovery

Notice that the model does not assume to be server based, as other deployment and implementation paradigms are possible.

Let us now consider how the container model addresses the architectural peculiarities identified in Section 1.1

Loose coupling Containers are independent one another and loosely coupled.

Message based interaction Messages are exchanged between the container and the service consumers (SD), on the one hand, and service directories (6), on the other hand.

Dynamic discovery The container exposes service descriptions and publishes them to service registries (SD).

Late binding Binding via the container interface can happen as late as execution time (SD).

Implementation neutrality and independency The container is defined independently of the service implementation.

Policy based behavior Policies and rules are internal to the container and drive its behavior (2,3,4).

Configurability Configuration is considered as being part of the container's characteristics (3).

Autonomicity Autonomicity is considered as being part of the container's characteristics (4).

Portability The container can be moved, providing the same service and needing, at most, only reconfiguration (3).

Granularity Containers can carry implementations of services at different granularity levels.

2 Open Research Issues

As we are in the early days of Service-Oriented Computing, the precise definition of service foundation can be merely considered an open research issues itself. However, it is not the only one. A number of related open questions sprout from the discussion. Most notably:

1. The identification of the requirements for the foundations/container. What is the optimal architecture for SOC? Can/should SOC become an extension of a the operating system? How does the service model depend on the specific consumers-providers peculiarities (scope, underlying HW/SW infrastructure, execution context)?
2. How to create the extensible architecture "mechanisms" for an extensible infrastructure. In particular, one needs a plug-in architecture to deal with extensible set of QoS properties. How does one deal with mechanisms in the various topics, such as policies for security, for transactions, for pricing, etc?
3. There is a need to support for agreements between service requester and consumer raising issues of negotiation, enforcement, and dynamic provisioning for compliance.

4. There is the need for new enabling technologies and architectures for existing IT infrastructures. What are these? (e.g. ESB, hardware appliances, Personal service bus?)
5. How can we achieve accuracy at the border of the container for the requirements of discovery? What should be the scope of service search? How deep should the search go into the container (scope)?
6. How to does the messaging mechanism satisfy the SOC desiderata?